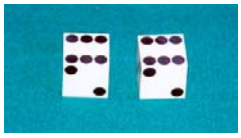
Advanced Machine Learning

Structured Models:

Hidden Markov Models versus Conditional Random Fields



Eric Xing

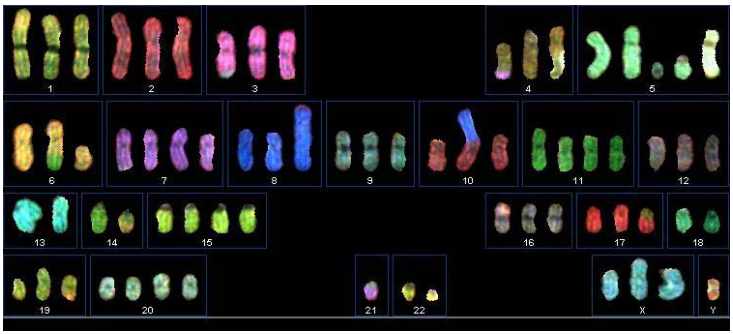
Lecture 9, August 12, 2009

Reading:

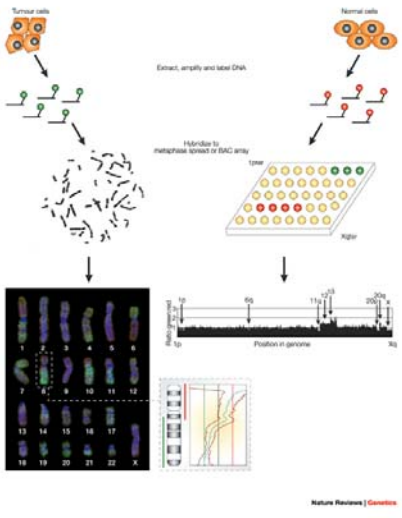
Dynamic clustering

Clustering Tumor Cell States

Chromosomes of tumor cell:

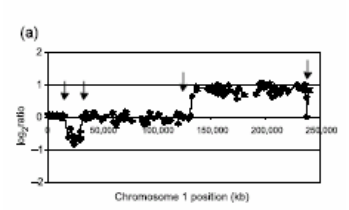


Array CGH (comparative genomic hybridization)



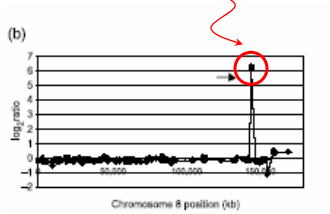
- The basic assumption of a CGH experiment is that the ratio of the binding of test and control DNA is proportional to the ratio of the copy numbers of sequences in the two samples.
- But various kinds of noises make the true observations less easy to interpret …

DNA Copy number aberration types in breast cancer

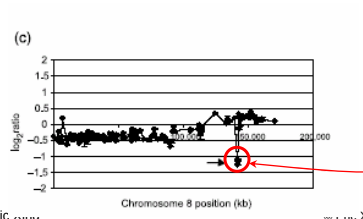


Copy number profile for chromosome 1 from 600 MPE cell line

60-70 fold amplification of CMYC region



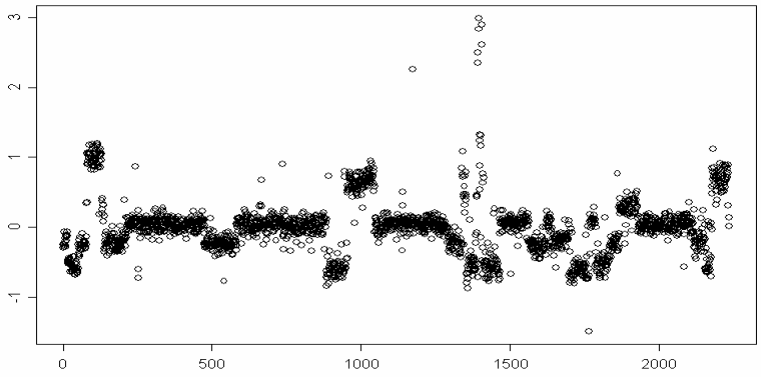
Copy number profile for chromosome 8 from COLO320 cell line



Copy number profile for chromosome 8 in MDA-MB-231 cell line

deletion

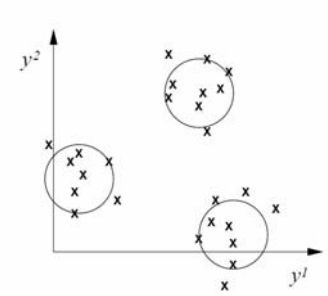
A real CGH run

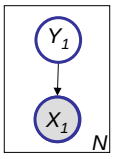


Out problem: how to cluster sequential data?

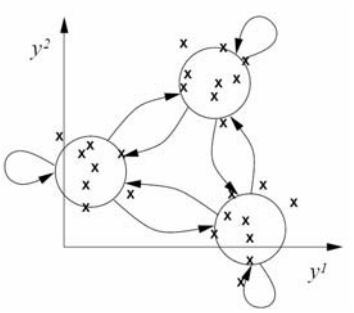
Hidden Markov Model:
from static to dynamic mixture models

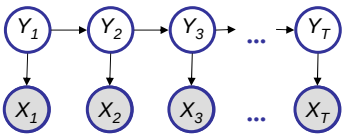
Static mixture





Dynamic mixture



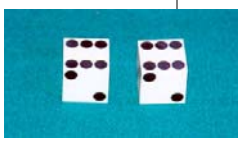


The Dishonest Casino

A casino has two dice:

- Fair die
  P(1) = P(2) = P(3) = P(5) = P(6) = 1/6
- Loaded die
  P(1) = P(2) = P(3) = P(5) = 1/10
  P(6) = 1/2

Casino player switches back-&-forth between fair and loaded die once every 20 turns

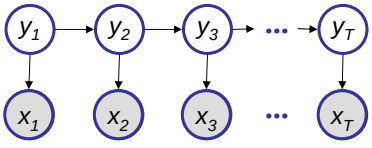


Game:

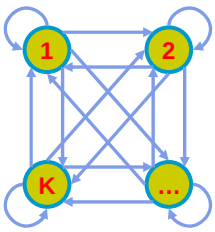
1. You bet $1
2. You roll (always with a fair die)
3. Casino player rolls (maybe with fair die, maybe with loaded die)
4. Highest number wins $2

Definition (of HMM)

- Observation space
  - **Alphabetic set:** $\mathbb{C} = \{c_1, c_2, \cdots, c_K\}$
  - **Euclidean space:** $\mathbb{R}^d$
- Index set of hidden states
  $\mathbb{I} = \{1,2,\cdots,M\}$
- Transition probabilities between any two states
  $p(y_t^j = 1 \mid y_{t-1}^i = 1) = a_{i,j},$
  or $p(y_t \mid y_{t-1}^i = 1) \sim \text{Multinomial}(a_{i,1}, a_{i,2}, \ldots, a_{i,M}), \forall i \in \mathbb{I}.$
- Start probabilities
  $p(y_1) \sim \text{Multinomial}(\pi_1, \pi_2, \ldots, \pi_M).$
- Emission probabilities associated with each state
  $p(x_t \mid y_t^i = 1) \sim \text{Multinomial}(b_{i,1}, b_{i,2}, \ldots, b_{i,K}), \forall i \in \mathbb{I}.$
  or in general:
  $p(x_t \mid y_t^i = 1) \sim \text{f}(\cdot \mid \theta_i), \forall i \in \mathbb{I}.$



Graphical model

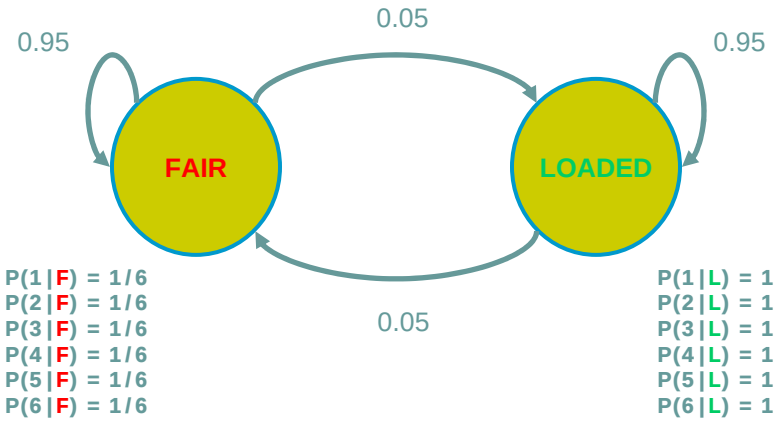


State automata

Applications of HMMs

- **Some early applications of HMMs**
  - finance, but we never saw them
  - speech recognition
  - modelling ion channels
- In the mid-late 1980s HMMs entered genetics and molecular biology, and they are now firmly entrenched.
- **Some current applications of HMMs to biology**
  - mapping chromosomes
  - aligning biological sequences
  - predicting sequence structure
  - inferring evolutionary relationships
  - finding genes in DNA sequence

The Dishonest Casino Model



Puzzles Regarding the Dishonest Casino

GIVEN: A sequence of rolls by the casino player

1245526462146146136136661664661636616366163616515615115146123562344

QUESTION

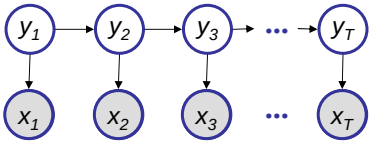
- How likely is this sequence, given our model of how the casino works?
  - This is the **EVALUATION** problem in HMMs
- What portion of the sequence was generated with the fair die, and what portion with the loaded die?
  - This is the **DECODING** question in HMMs
- How "loaded" is the loaded die? How "fair" is the fair die? How often does the casino player change from fair to loaded, and back?
  - This is the **LEARNING** question in HMMs

Joint Probability

1245526462146146136136661664661636616366163616515615115146123562344

Probability of a Parse

- Given a sequence $\mathbf{x} = x_1 \ldots\ldots x_T$ and a parse $\mathbf{y} = y_1, \ldots\ldots, y_T$,
- To find how likely is the parse: (given our HMM and the sequence)



$p(\mathbf{x}, \mathbf{y}) = p(x_1 \ldots\ldots x_T, y_1, \ldots\ldots, y_T)$ (Joint probability)

$= p(y_1)\, p(x_1 \mid y_1)\, p(y_2 \mid y_1)\, p(x_2 \mid y_2) \ldots p(y_T \mid y_{T-1})\, p(x_T \mid y_T)$

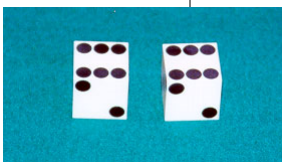
$= p(y_1)\, P(y_2 \mid y_1) \ldots p(y_T \mid y_{T-1}) \times p(x_1 \mid y_1)\, p(x_2 \mid y_2) \ldots p(x_T \mid y_T)$

- Marginal probability: $p(\mathbf{x}) = \sum_{\mathbf{y}} p(\mathbf{x}, \mathbf{y}) = \sum_{y_1} \sum_{y_2} \cdots \sum_{y_N} \pi_{y_1} \prod_{t=2}^{T} a_{y_{t-1}, y_t} \prod_{t=1}^{T} p(x_t \mid y_t)$
- Posterior probability: $p(\mathbf{y} \mid \mathbf{x}) = p(\mathbf{x}, \mathbf{y}) / p(\mathbf{x})$

Example: the Dishonest Casino

- Let the sequence of rolls be:
  - x = 1, 2, 1, 5, 6, 2, 1, 6, 2, 4
- Then, what is the likelihood of
  - y = Fair, Fair, Fair, Fair, Fair, Fair, Fair, Fair, Fair, Fair?

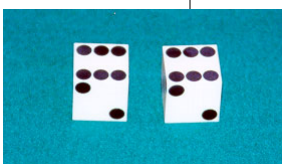
  (say initial probs $a_{0Fair} = ½$, $a_{0Loaded} = ½$)



½ × P(1 | Fair) P(Fair | Fair) P(2 | Fair) P(Fair | Fair) … P(4 | Fair) =

$½ \times (\mathbf{1/6})^{10} \times (\mathbf{0.95})^{9} = .00000000521158647211 = 5.21 \times 10^{-9}$

Example: the Dishonest Casino



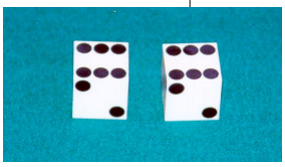
- So, the likelihood the die is fair in all this run is just 5.21×10^{-9}
- OK, but what is the likelihood of
  - π = Loaded, Loaded, Loaded, Loaded, Loaded, Loaded, Loaded, Loaded, Loaded, Loaded?

$\frac{1}{2} \times P(1 \mid \text{Loaded})\, P(\text{Loaded} \mid \text{Loaded}) \ldots P(4 \mid \text{Loaded}) =$

$\frac{1}{2} \times (\mathbf{1/10})^8 \times (\mathbf{1/2})^2\, (\mathbf{0.95})^9 = .00000000078781176215 = 0.79 \times 10^{-9}$

- Therefore, it is after all 6.59 times more likely that the die is fair all the way, than that it is loaded all the way

Example: the Dishonest Casino



- Let the sequence of rolls be:
  - x = 1, 6, 6, 5, 6, 2, 6, 6, 3, 6
- Now, what is the likelihood π = F, F, …, F?
  - $\frac{1}{2} \times (1/6)^{10} \times (0.95)^9 = 0.5 \times 10^{-9}$, same as before
- What is the likelihood $\boldsymbol{y}$ = L, L, …, L?

$\frac{1}{2} \times (1/10)^4 \times (1/2)^6\, (0.95)^9 = .00000049238235134735 = 5 \times 10^{-7}$

- So, it is 100 times more likely the die is loaded

Three Main Questions on HMMs

1. Evaluation

   GIVEN an HMM $\mathcal{M}$, and a sequence $\boldsymbol{x}$,
   FIND Prob ($\boldsymbol{x}$ | $\mathcal{M}$)
   ALGO. Forward

2. Decoding

   GIVEN an HMM $\mathcal{M}$, and a sequence $\boldsymbol{x}$,
   FIND the sequence $\mathbf{y}$ of states that maximizes, e.g., P($\boldsymbol{y}$ | $\boldsymbol{x}$, $\mathcal{M}$), or the most probable subsequence of states
   ALGO. Viterbi, Forward-backward

3. Learning

   GIVEN an HMM $\boldsymbol{M}$, with unspecified transition/emission probs., and a sequence $\boldsymbol{x}$,
   FIND parameters $\theta = (\pi_i, a_{ij}, \eta_{ik})$ that maximize $P(\boldsymbol{x} | \theta)$
   ALGO. Baum-Welch (EM)

The Forward Algorithm

- We want to calculate $P(\mathbf{x})$, the likelihood of $\mathbf{x}$, given the HMM
  - Sum over all possible ways of generating $\mathbf{x}$:

$$p(\mathbf{x}) = \sum_{\mathbf{y}} p(\mathbf{x}, \mathbf{y}) = \sum_{y_1} \sum_{y_2} \cdots \sum_{y_N} \pi_{y_1} \prod_{t=2}^{T} a_{y_{t-1}, y_t} \prod_{t=1}^{T} p(x_t \mid y_t)$$

  - To avoid summing over an exponential number of paths $\mathbf{y}$, define

$$\alpha(y_t^k = 1) = \alpha_t^k \overset{\text{def}}{=} P(x_1, \ldots, x_t, y_t^k = 1)$$ (the forward probability)

  - The recursion:

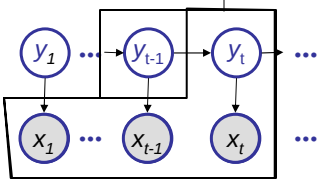
$$\alpha_t^k = p(x_t \mid y_t^k = 1) \sum_i \alpha_{t-1}^i a_{i,k}$$

$$P(\mathbf{x}) = \sum_k \alpha_T^k$$

The Forward Algorithm – derivation

- Compute the forward probability:

$$\alpha_t^k = P(x_1,\ldots,x_{t-1},x_t,y_t^k=1)$$



$$= \sum_{y_{t-1}} P(x_1,\ldots,x_{t-1},y_{t-1})P(y_t^k=1\mid y_{t-1},x_1,\ldots,x_{t-1})P(x_t\mid y_t^k=1,x_1,\ldots,x_{t-1},y_{t-1})$$
$$= \sum_{y_{t-1}} P(x_1,\ldots,x_{t-1},y_{t-1})P(y_t^k=1\mid y_{t-1})P(x_t\mid y_t^k=1)$$
$$= P(x_t\mid y_t^k=1)\sum_i P(x_1,\ldots,x_{t-1},y_{t-1}^i=1)P(y_t^k=1\mid y_{t-1}^i=1)$$
$$= P(x_t\mid y_t^k=1)\sum_i \alpha_{t-1}^i a_{i,k}$$

Chain rule: $P(A,B,C) = P(A)P(B\mid A)P(C\mid A,B)$

The Forward Algorithm

- We can compute α_t^k for all k, t, using dynamic programming!

Initialization:

$$\alpha_1^k = P(x_1\mid y_1^k=1)\pi_k$$

$$\alpha_1^k = P(x_1,y_1^k=1)$$
$$= P(x_1\mid y_1^k=1)P(y_1^k=1)$$
$$= P(x_1\mid y_1^k=1)\pi_k$$

Iteration:

$$\alpha_t^k = P(x_t\mid y_t^k=1)\sum_i \alpha_{t-1}^i a_{i,k}$$

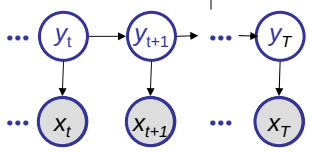
Termination:

$$P(\mathbf{x}) = \sum_k \alpha_T^k$$

The Backward Algorithm

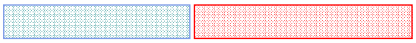
- We want to compute $P(y_t^k = 1 \mid \mathbf{x})$,

  the posterior probability distribution on the t^{th} position, given $\mathbf{x}$



  - We start by computing

$$P(y_t^k = 1, \mathbf{x}) = P(x_1,\ldots,x_t, y_t^k = 1, x_{t+1},\ldots,x_T)$$
$$= P(x_1,\ldots,x_t, y_t^k = 1)P(x_{t+1},\ldots,x_T \mid x_1,\ldots,x_t, y_t^k = 1)$$
$$= P(x_1\ldots x_t, y_t^k = 1)P(x_{t+1}\ldots x_T \mid y_t^k = 1)$$



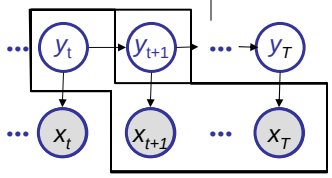
Forward, α_t^k **Backward,** $\beta_t^k = P(x_{t+1},\ldots,x_T \mid y_t^k = 1)$

  - The recursion:

$$\beta_t^k = \sum_i a_{k,i}\, p(x_{t+1} \mid y_{t+1}^i = 1)\beta_{t+1}^i$$

The Backward Algorithm – derivation

- Define the backward probability:



$$\beta_t^k = P(x_{t+1},\ldots,x_T \mid y_t^k = 1)$$
$$= \sum_{y_{t+1}} P(x_{t+1},\ldots,x_T, y_{t+1} \mid y_t^k = 1)$$
$$= \sum_i P(y_{t+1}^i = 1 \mid y_t^k = 1)p(x_{t+1} \mid y_{t+1}^i = 1, y_t^k = 1)P(x_{t+2},\ldots,x_T \mid x_{t+1}, y_{t+1}^i = 1, y_t^k = 1)$$
$$= \sum_i P(y_{t+1}^i = 1 \mid y_t^k = 1)p(x_{t+1} \mid y_{t+1}^i = 1)P(x_{t+2},\ldots,x_T \mid y_{t+1}^i = 1)$$
$$= \sum_i a_{k,i}\, p(x_{t+1} \mid y_{t+1}^i = 1)\beta_{t+1}^i$$

Chain rule: $P(A,B,C \mid \alpha) = P(A \mid \alpha)P(B \mid A,\alpha)P(C \mid A,B,\alpha)$

The Backward Algorithm

- We can compute β_t^k for all k, t, using dynamic programming!

Initialization:

$$\beta_T^k = 1, \ \forall k$$

Iteration:

$$\beta_t^k = \sum_i a_{k,i} P(x_{t+1} \mid y_{t+1}^i = 1) \beta_{t+1}^i$$

Termination:

$$P(\mathbf{x}) = \sum_k \alpha_1^k \beta_1^k$$

Example:

x = 1, 2, 1, 5, 6, 2, 1, 6, 2, 4

P(1|F) = 1/6	P(1|L) = 1/10
P(2|F) = 1/6	P(2|L) = 1/10
P(3|F) = 1/6	P(3|L) = 1/10
P(4|F) = 1/6	P(4|L) = 1/10
P(5|F) = 1/6	P(5|L) = 1/10
P(6|F) = 1/6	P(6|L) = 1/2

$$\alpha_t^k = P(x_t \mid y_t^k = 1) \sum_i \alpha_{t-1}^i a_{i,k}$$

$$\beta_t^k = \sum_i a_{k,i} P(x_{t+1} \mid y_{t+1}^i = 1) \beta_{t+1}^i$$

Posterior decoding

- We can now calculate

$$P(y_t^k = 1 \mid \mathbf{x}) = \frac{P(y_t^k = 1, \mathbf{x})}{P(\mathbf{x})} = \frac{\alpha_t^k \beta_t^k}{P(\mathbf{x})}$$

- Then, we can ask
  - What is the most likely state at position *t* of sequence $\mathbf{x}$:

$$k_t^* = \arg\max_k P(y_t^k = 1 \mid \mathbf{x})$$

  - Note that this is an MPA of a single hidden state, what if we want to a MPA of a whole hidden state sequence?
  - Posterior Decoding: $\left\{ y_t^{k_t^*} = 1 : t = 1 \cdots T \right\}$
  - This is different from MPA of a whole sequence states
  - This can be understood as *bit error rate* vs. *word error rate*

Example:
MPA of *X* ?
MPA of *(X, Y)* ?

of hidden

x	y	P(x,y)
0	0	0.35
0	1	0.05
1	0	0.3
1	1	0.3

Viterbi decoding

- GIVEN $\mathbf{x} = x_1, \ldots, x_T$, we want to find $\mathbf{y} = y_1, \ldots, y_T$, such that $P(\mathbf{y}|\mathbf{x})$ is maximized:

$$\mathbf{y}^* = \text{argmax}_{\mathbf{y}} P(\mathbf{y} \mid \mathbf{x}) = \text{argmax}_{\pi} P(\mathbf{y}, \mathbf{x})$$

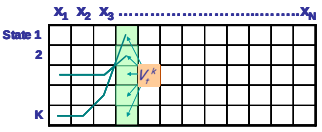
  - Let

$$V_t^k = \max_{\{y_1, \ldots, y_{t-1}\}} P(x_1, \ldots, x_{t-1}, y_1, \ldots, y_{t-1}, x_t, y_t^k = 1)$$

  = Probability of most likely **sequence of states** ending at state $y_t = k$

  - The recursion:

$$V_t^k = p(x_t \mid y_t^k = 1) \max_i a_{i,k} V_{t-1}^i$$



  - Underflows are a significant problem

$$p(x_1, \ldots, x_t, y_1, \ldots, y_t) = \pi_{y_1} a_{y_1, y_2} \cdots a_{y_{t-1}, y_t} b_{y_1, x_1} \cdots b_{y_t, x_t}$$

    - These numbers become extremely small – underflow
    - Solution: Take the logs of all values: $V_t^k = \log p(x_t \mid y_t^k = 1) + \max_i \left(\log(a_{i,k}) + V_{t-1}^i \right)$

Computational Complexity and implementation details

- What is the running time, and space required, for Forward, and Backward?

$$\alpha_t^k = p(x_t \mid y_t^k = 1) \sum_i \alpha_{t-1}^i a_{i,k}$$

$$\beta_t^k = \sum_i a_{k,i}\, p(x_{t+1} \mid y_{t+1}^i = 1) \beta_{t+1}^i$$

$$V_t^k = p(x_t \mid y_t^k = 1) \max_i a_{i,k} V_{t-1}^i$$

Time: $O(K^2N)$; Space: $O(KN)$.

- Useful implementation technique to avoid underflows
  - Viterbi: sum of logs
  - Forward/Backward: rescaling at each position by multiplying by a constant

Learning HMM: two scenarios

- **Supervised learning**: estimation when the "right answer" is known
  - **Examples:**

    GIVEN: a genomic region $x = x_1 \ldots x_{1,000,000}$ where we have good (experimental) annotations of the CpG islands

    GIVEN: the casino player allows us to observe him one evening, as he changes dice and produces 10,000 rolls

- **Unsupervised learning**: estimation when the "right answer" is unknown
  - **Examples:**

    GIVEN: the porcupine genome; we don't know how frequent are the CpG islands there, neither do we know their composition

    GIVEN: 10,000 rolls of the casino player, but we don't see when he changes dice

- **QUESTION:** Update the parameters θ of the model to maximize $P(x|\theta)$ --- Maximal likelihood (ML) estimation

Supervised ML estimation

- Given $x = x_1 \ldots x_N$ for which the true state path $y = y_1 \ldots y_N$ is known,
  - **Define:**

    A_{ij} = # times state transition $i \to j$ occurs in $\mathbf{y}$

    B_{ik} = # times state i in $\mathbf{y}$ emits k in $\mathbf{x}$

  - **We can show that the maximum likelihood parameters θ are:**

$$a_{ij}^{ML} = \frac{\#(i \to j)}{\#(i \to \bullet)} = \frac{\sum_n \sum_{t=2}^{T} y_{n,t-1}^{i} y_{n,t}^{j}}{\sum_n \sum_{t=2}^{T} y_{n,t-1}^{i}} = \frac{A_{ij}}{\sum_{j'} A_{ij'}}$$

$$b_{ik}^{ML} = \frac{\#(i \to k)}{\#(i \to \bullet)} = \frac{\sum_n \sum_{t=1}^{T} y_{n,t}^{i} x_{n,t}^{k}}{\sum_n \sum_{t=1}^{T} y_{n,t}^{i}} = \frac{B_{ik}}{\sum_{k'} B_{ik'}}$$

(Homework!)

  - **What if y is continuous? We can treat $\{(x_{n,t}, y_{n,t}) : t = 1:T, n = 1:N\}$ as $N \times T$ observations of, e.g., a Gaussian, and apply learning rules for Gaussian ...**

(Homework!)

Pseudocounts

- Solution for small training sets:
  - Add pseudocounts

    A_{ij} = # times state transition $i \to j$ occurs in $\mathbf{y}$ + R_{ij}

    B_{ik} = # times state i in $\mathbf{y}$ emits k in $\mathbf{x}$ + S_{ik}

  - R_{ij}, S_{ij} are pseudocounts representing our prior belief
  - Total pseudocounts: $R_i = \Sigma_j R_{ij}$, $S_i = \Sigma_k S_{ik}$,
    - --- "strength" of prior belief,
    - --- total number of imaginary instances in the prior
- Larger total pseudocounts $\Rightarrow$ strong prior belief
- Small total pseudocounts: just to avoid 0 probabilities --- smoothing

Unsupervised ML estimation

- Given $x = x_1 \ldots x_N$ for which the true state path $y = y_1 \ldots y_N$ is unknown,
  - **EXPECTATION MAXIMIZATION**
  0. Starting with our best guess of a model M, parameters θ:
  1. Estimate A_{ij} , B_{ik} in the training data
     - How? $A_{ij} = \sum_{n,t} \langle y^i_{n,t-1} y^j_{n,h} \rangle$ $B_{ik} = \sum_{n,t} \langle y^i_{n,t} \rangle x^k_{n,t}$, How? (homework)
  2. Update θ according to A_{ij} , B_{ik}
     - Now a "supervised learning" problem
  3. Repeat 1 & 2, until convergence

  This is called the Baum-Welch Algorithm

  We can get to a provably more (or equally) likely parameter set θ each iteration

The Baum Welch algorithm

- The complete log likelihood

$$\ell_c(\boldsymbol{\theta}; \mathbf{x}, \mathbf{y}) = \log p(\mathbf{x}, \mathbf{y}) = \log \prod_n \left(p(y_{n,1}) \prod_{t=2}^{T} p(y_{n,t} \mid y_{n,t-1}) \prod_{t=1}^{T} p(x_{n,t} \mid x_{n,t}) \right)$$

- The expected complete log likelihood

$$\langle \ell_c(\boldsymbol{\theta}; \mathbf{x}, \mathbf{y}) \rangle = \sum_n \left(\langle y^i_{n,1} \rangle_{p(y_{n,1} \mid \mathbf{x}_n)} \log \pi_i \right) + \sum_n \sum_{t=2}^{T} \left(\langle y^i_{n,t-1} y^j_{n,t} \rangle_{p(y_{n,t-1}, y_{n,t} \mid \mathbf{x}_n)} \log a_{i,j} \right) + \sum_n \sum_{t=1}^{T} \left(x^k_{n,t} \langle y^i_{n,t} \rangle_{p(y_{n,t} \mid \mathbf{x}_n)} \log b_{i,k} \right)$$

- EM
  - The **E** step

$$\gamma^i_{n,t} = \langle y^i_{n,t} \rangle = p(y^i_{n,t} = 1 \mid \mathbf{x}_n)$$

$$\xi^{i,j}_{n,t} = \langle y^i_{n,t-1} y^j_{n,t} \rangle = p(y^i_{n,t-1} = 1, y^j_{n,t} = 1 \mid \mathbf{x}_n)$$

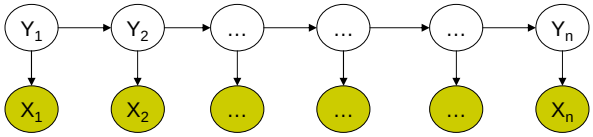
  - The **M** step ("symbolically" identical to MLE)

$$\pi_i^{ML} = \frac{\sum_n \gamma^i_{n,1}}{N} \qquad a_{ij}^{ML} = \frac{\sum_n \sum_{t=2}^{T} \xi^{i,j}_{n,t}}{\sum_n \sum_{t=1}^{T-1} \gamma^i_{n,t}} \qquad b_{ik}^{ML} = \frac{\sum_n \sum_{t=1}^{T} \gamma^i_{n,t} x^k_{n,t}}{\sum_n \sum_{t=1}^{T-1} \gamma^i_{n,t}}$$

Summary

- Modeling hidden transitional trajectories (in discrete state space, such as cluster label, DNA copy number, dice-choice, etc.) underlying observed sequence data (discrete, such as dice outcomes; or continuous, such as CGH signals)
- Useful for parsing, segmenting sequential data
- Important HMM computations:
  - The joint likelihood of a parse and data can be written as a product to local terms (i.e., initial prob, transition prob, emission prob.)
  - Computing marginal likelihood of the observed sequence: forward algorithm
  - Predicting a single hidden state: forward-backward
  - Predicting an entire sequence of hidden states: viterbi
  - Learning HMM parameters: an EM algorithm known as Baum-Welch

Shortcomings of Hidden Markov Model

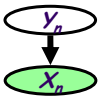


- HMM models capture dependences between each state and only its corresponding observation
  - NLP example: In a sentence segmentation task, each segmental state may depend not just on a single word (and the adjacent segmental stages), but also on the (non-local) features of the whole line such as line length, indentation, amount of white space, etc.
- Mismatch between learning objective function and prediction objective function
  - HMM learns a joint distribution of states and observations P(**Y**, **X**), but in a prediction task, we need the conditional probability P(**Y**|**X**)

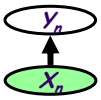
Recall Generative vs. Discriminative Classifiers

- Goal: Wish to learn f: X $\rightarrow$ Y, e.g., P(Y|X)

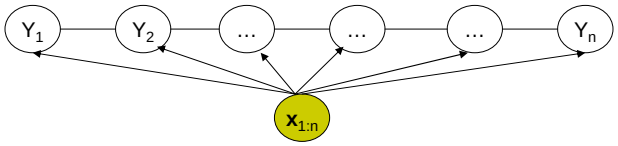
- Generative classifiers (e.g., Naïve Bayes):
  - Assume some functional form for P(X|Y), P(Y)
    This is a '***generative***' model of the data!
  - Estimate parameters of P(X|Y), P(Y) directly from training data
  - Use Bayes rule to calculate P(Y|X= x)



- Discriminative classifiers (e.g., logistic regression)
  - Directly assume some functional form for P(Y|X)
    This is a '***discriminative***' model of the data!
  - Estimate parameters of P(Y|X) directly from training data



Conditional Random Fields

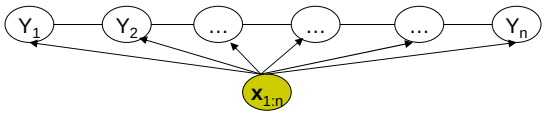


$$P(\mathbf{y}_{1:n}|\mathbf{x}_{1:n}) = \frac{1}{Z(\mathbf{x}_{1:n})}\prod_{i=1}^{n}\phi(y_i, y_{i-1}, \mathbf{x}_{1:n}) = \frac{1}{Z(\mathbf{x}_{1:n}, \mathbf{w})}\prod_{i=1}^{n}\exp(\mathbf{w}^T\mathbf{f}(y_i, y_{i-1}, \mathbf{x}_{1:n}))$$

- CRF is a partially directed model
  - Discriminative model
  - Usage of global normalizer Z(**x**)
  - Models the dependence between each state and the entire observation sequence

Conditional Random Fields

- General parametric form:



$$P(\mathbf{y}|\mathbf{x}) = \frac{1}{Z(\mathbf{x},\lambda,\mu)} \exp\Big(\sum_{i=1}^{n}\Big(\sum_{k}\lambda_k f_k(y_i,y_{i-1},\mathbf{x}) + \sum_{l}\mu_l g_l(y_i,\mathbf{x})\Big)\Big)$$

$$= \frac{1}{Z(\mathbf{x},\lambda,\mu)} \exp\Big(\sum_{i=1}^{n}\big(\lambda^T \mathbf{f}(y_i,y_{i-1},\mathbf{x}) + \mu^T \mathbf{g}(y_i,\mathbf{x})\big)\Big)$$

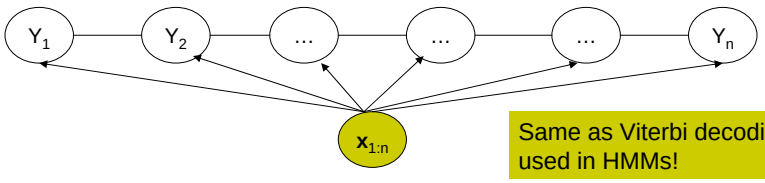
$$\text{where } Z(\mathbf{x},\lambda,\mu) = \sum_{\mathbf{y}} \exp\Big(\sum_{i=1}^{n}\big(\lambda^T \mathbf{f}(y_i,y_{i-1},\mathbf{x}) + \mu^T \mathbf{g}(y_i,\mathbf{x})\big)\Big)$$

CRFs: Inference

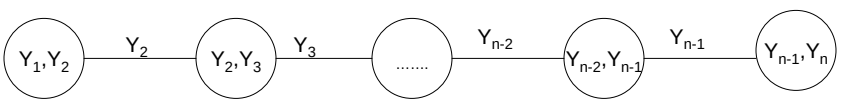
- Given CRF parameters λ and μ, find the $\mathbf{y}^*$ that maximizes $P(\mathbf{y}|\mathbf{x})$

$$\mathbf{y}^* = \arg\max_{\mathbf{y}} \exp\Big(\sum_{i=1}^{n}\big(\lambda^T \mathbf{f}(y_i,y_{i-1},\mathbf{x}) + \mu^T \mathbf{g}(y_i,\mathbf{x})\big)\Big)$$

  - Can ignore $Z(\mathbf{x})$ because it is not a function of $\mathbf{y}$
- Run the max-product algorithm on the junction-tree of CRF:



Same as Viterbi decoding used in HMMs!



CRF learning

- Given $\{(\mathbf{x}_d, \mathbf{y}_d)\}_{d=1}^N$, find λ^*, μ^* such that

$$\begin{aligned}\lambda*, \mu* &= \arg\max_{\lambda,\mu} L(\lambda,\mu) = \arg\max_{\lambda,\mu} \prod_{d=1}^{N} P(\mathbf{y}_d|\mathbf{x}_d,\lambda,\mu) \\ &= \arg\max_{\lambda,\mu} \prod_{d=1}^{N} \frac{1}{Z(\mathbf{x}_d,\lambda,\mu)} \exp\Big(\sum_{i=1}^{n}(\lambda^T \mathbf{f}(y_{d,i}, y_{d,i-1}, \mathbf{x}_d) + \mu^T \mathbf{g}(y_{d,i}, \mathbf{x}_d))\Big) \\ &= \arg\max_{\lambda,\mu} \sum_{d=1}^{N}\Big(\sum_{i=1}^{n}(\lambda^T \mathbf{f}(y_{d,i}, y_{d,i-1}, \mathbf{x}_d) + \mu^T \mathbf{g}(y_{d,i}, \mathbf{x}_d)) - \log Z(\mathbf{x}_d,\lambda,\mu)\Big)\end{aligned}$$

- Computing the gradient w.r.t λ:

Gradient of the log-partition function in an exponential family is the expectation of the sufficient statistics.

$$\nabla_\lambda L(\lambda,\mu) = \sum_{d=1}^{N}\Big(\sum_{i=1}^{n} \mathbf{f}(y_{d,i}, y_{d,i-1}, \mathbf{x}_d) - \sum_{\mathbf{y}}\Big(P(\mathbf{y}|\mathbf{x}_\mathbf{d}) \sum_{i=1}^{n} \mathbf{f}(y_{d,i}, y_{d,i-1}, \mathbf{x}_d)\Big)\Big)$$

CRF learning

$$\nabla_\lambda L(\lambda,\mu) = \sum_{d=1}^{N}\Big(\sum_{i=1}^{n} \mathbf{f}(y_{d,i}, y_{d,i-1}, \mathbf{x}_d) - \sum_{\mathbf{y}}\Big(P(\mathbf{y}|\mathbf{x}_\mathbf{d}) \sum_{i=1}^{n} \mathbf{f}(y_i, y_{i-1}, \mathbf{x}_d)\Big)\Big)$$

- Computing the model expectations:
  - Requires exponentially large number of summations: Is it intractable?

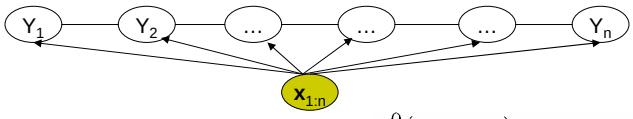
$$\begin{aligned}\sum_{\mathbf{y}}\Big(P(\mathbf{y}|\mathbf{x}_d) \sum_{i=1}^{n} \mathbf{f}(y_i, y_{i-1}, \mathbf{x}_d)\Big) &= \sum_{i=1}^{n}\Big(\sum_{\mathbf{y}} \mathbf{f}(y_i, y_{i-1}, \mathbf{x}_d) P(\mathbf{y}|\mathbf{x}_d)\Big) \\ &= \sum_{i=1}^{n} \sum_{y_i, y_{i-1}} \mathbf{f}(y_i, y_{i-1}, \mathbf{x}_d) P(y_i, y_{i-1}|\mathbf{x}_d)\end{aligned}$$

Expectation of **f** over the corresponding marginal probability of neighboring nodes!!

- Tractable!
  - Can compute marginals using the sum-product algorithm on the chain

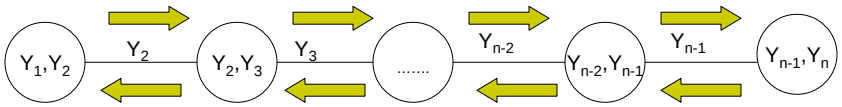
CRF learning

- Computing marginals using junction-tree calibration:



- Junction Tree Initialization:

$$\alpha^0(y_i, y_{i-1}) = \exp(\lambda^T \mathbf{f}(y_i, y_{i-1}, \mathbf{x}_d) + \mu^T \mathbf{g}(y_i, \mathbf{x}_d))$$



- After calibration:

Also called forward-backward algorithm

$$P(y_i, y_{i-1}|\mathbf{x}_d) \propto \alpha(y_i, y_{i-1})$$
$$\Rightarrow P(y_i, y_{i-1}|\mathbf{x}_d) = \frac{\alpha(y_i, y_{i-1})}{\sum_{y_i, y_{i-1}} \alpha(y_i, y_{i-1})} = \alpha'(y_i, y_{i-1})$$

CRF learning

- Computing feature expectations using calibrated potentials:

$$\sum_{y_i, y_{i-1}} \mathbf{f}(y_i, y_{i-1}, \mathbf{x}_d) P(y_i, y_{i-1}|\mathbf{x}_d) = \sum_{y_i, y_{i-1}} \mathbf{f}(y_i, y_{i-1}, \mathbf{x}_d) \alpha'(y_i, y_{i-1})$$

- Now we know how to compute $\nabla_\lambda L(\lambda,\mu)$:

$$\nabla_\lambda L(\lambda, \mu) = \sum_{d=1}^{N} (\sum_{i=1}^{n} \mathbf{f}(y_{d,i}, y_{d,i-1}, \mathbf{x}_d) - \sum_{\mathbf{y}} (P(\mathbf{y}|\mathbf{x_d}) \sum_{i=1}^{n} \mathbf{f}(y_i, y_{i-1}, \mathbf{x}_d)))$$
$$= \sum_{d=1}^{N} (\sum_{i=1}^{n} (\mathbf{f}(y_{d,i}, y_{d,i-1}, \mathbf{x}_d) - \sum_{y_i, y_{i-1}} \alpha'(y_i, y_{i-1}) \mathbf{f}(y_i, y_{i-1}, \mathbf{x}_d)))$$

- Learning can now be done using gradient ascent:

$$\lambda^{(t+1)} = \lambda^{(t)} + \eta \nabla_\lambda L(\lambda^{(t)}, \mu^{(t)})$$
$$\mu^{(t+1)} = \mu^{(t)} + \eta \nabla_\mu L(\lambda^{(t)}, \mu^{(t)})$$

CRF learning

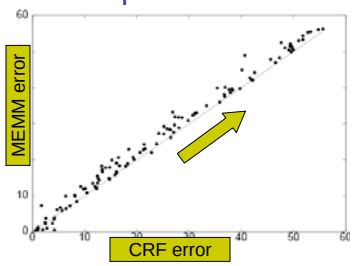
- In practice, we use a Gaussian Regularizer for the parameter vector to improve generalizability

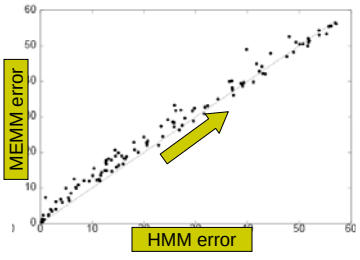
$$\lambda*, \mu* \quad = \quad \arg\max_{\lambda,\mu} \sum_{d=1}^{N} \log P(\mathbf{y}_d|\mathbf{x}_d, \lambda, \mu) - \frac{1}{2\sigma^2}(\lambda^T\lambda + \mu^T\mu)$$

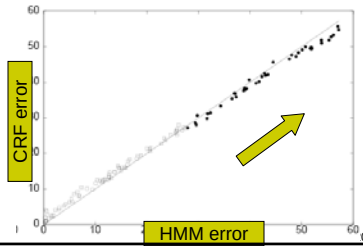
- In practice, gradient ascent has very slow convergence
  - Alternatives:
    - Conjugate Gradient method
    - Limited Memory Quasi-Newton Methods

CRFs: some empirical results

- Comparison of error rates on synthetic data







Data is increasingly higher order in the direction of arrow

CRFs achieve the lowest error rate for higher order data

CRFs: some empirical results

- Parts of Speech tagging

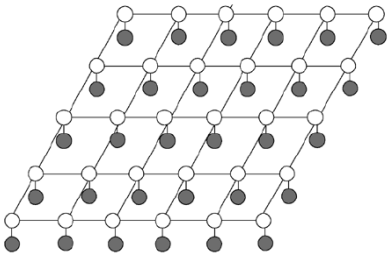
model	*error*	*oov error*
HMM	5.69%	45.99%
MEMM	6.37%	54.61%
CRF	5.55%	48.05%
MEMM$^+$	4.81%	26.99%
CRF$^+$	4.27%	23.76%

$^+$Using spelling features

  - Using same set of features: HMM >=< CRF > MEMM
  - Using additional overlapping features: CRF$^+$ > MEMM$^+$ >> HMM

Other CRFs

- So far we have discussed only 1-dimensional chain CRFs
  - Inference and learning: exact
- We could also have CRFs for arbitrary graph structure
  - E.g: Grid CRFs
  - Inference and learning no longer tractable
  - Approximate techniques used
    - MCMC Sampling
    - Variational Inference
    - Loopy Belief Propagation
  - We will discuss these techniques in the future



Summary

- Conditional Random Fields are partially directed discriminative models
- Inference for 1-D chain CRFs is exact
  - Same as Max-product or Viterbi decoding
- Learning also is exact
  - globally optimum parameters can be learned
  - Requires using sum-product or forward-backward algorithm
- CRFs involving arbitrary graph structure are intractable in general
  - E.g.: Grid CRFs
  - Inference and learning require approximation techniques
    - MCMC sampling
    - Variational methods
    - Loopy BP